About the Legacy Version

This document describes the "legacy" version of the Resource Compiler (RC16). RC16 is an **obsolete** level of the Resource Compiler. It remains available for applications that cannot utilize the newer (stricter) level of RC.

The new version of RC (that replaces RC16) is described in the online *Tools Reference* document.

Resource Compiler

The OS/2 Resource Compiler (RC) is an application-development tool that lets you add application resources, such as message strings, pointers, menus, and dialog boxes, to the executable file of your application. The Resource Compiler is primarily intended to prepare data for OS/2 applications that use functions such as WinLoadString, WinLoadPointer, WinLoadMenu, and WinLoadDlg. These functions load resources from the executable file of your application or another specified executable file. The application can then use the loaded resources as needed.

The Resource Compiler and the resource functions let you quickly define and/or modify application resources without recompiling the application itself. That is, RC can modify the resources in an executable file at any time without affecting the rest of the file. This means that you can create custom applications from a single executable file - you just use RC to add the custom resources you need to each application.

The Resource Compiler is especially important for international applications because it lets you define all language-dependent data, such as message strings, as resources. Preparing the application for a new language is simply a matter of adding new resources to the existing executable file.

Note: Make sure the file RCPP.EXE (the Resource Compiler preprocessor) is available for the use of the Resource Compiler. It can be in the current directory, or in a directory to which there is a path.

Command-Line Options

The following options can be specified on the Resource Compiler command line:

-d <defname>[=<value>]	Define macro to preprocessor
-i <pathspec>	Include file path
-n	Suppress the display of the logo and the copyright information
-r	Create .res file
-p	Pack - 386 resources will not cross 64K boundaries.
-cp (or -k) {<codepage>|<lbs,lbe>...}	DBCS code page or lead byte information
-x[{1|2}]	Exepack - compress resources, using method 1 or 2.
-cc <countrycode>	Country code
-w2	Suppress the display of all warning and informational messages. Errors and fatal errors continue to be displayed.
-h (or -?)	Access Help

Leave a blank after the letter when using option -cc, -d, -i , -cp or -k. Upper- or lowercase letters can be used.

Explanation of Command-Line Options

The -d option is useful for passing conditional-compilation flags to the preprocessor. The <defname> is a sequence of letters, underscore symbols, and digits which does not begin with a digit. The <value> is a sequence of symbols which you want to substitute for the <defname> wherever it appears in the input script file. If you omit the =<value>, the <defname> will be set to the default value 1. For example, the option -d _3d is equivalent to including at the beginning of the input file this line:

```
#define   _3d      1
```

You can use the -d option up to 8 times to define different macros from the command line.

The -i option defines paths for files to be included with the source file. The <pathspec> is any path where you want RC to search for files included by the preprocessor #include directive. The <pathspec> must not contain embedded blanks. To include more than one path, code the -i option once for each path. The preprocessor reads paths from the INCLUDE environment variable after reading the paths you provide with -i options.

The -r option will create in your current directory a binary resource file containing the resources you compile. The -r option takes no argument. The name given to this binary resource file will be the same as the name of the input resource script file except that the extension will be .RES instead of .RC. When you use -r, you do not bind resources to an executable file.

The -p option is used only when binding resources to an executable. It positions resources so that they do not cross 64K boundaries.

The -cp or -k option is used to specify code page information for the resource script file to be compiled. The <codepage> is a numeric code page value. For a list of code page values, see the code page table under COUNTRYCODE in the online book *OS/2 Warp Control Program Programming Reference* . Instead of specifying a code page, you may provide a sequence of pairs of DBCS lead byte code points. Each pair of numbers gives the lower and upper limit of a range of code points which are to be interpreted as DBCS lead bytes (see DBCS Code Pages and Country Codes for more information). The code page must be valid for the country code in effect: either the default country code or the country specified using the -cc option.

The -cc option allows you to specify a country code for the resource script file to be compiled. The <countrycode> is a number from the table under COUNTRYCODE in the online book *OS/2 Warp Control Program Programming Reference* .

The -x option is used only when binding resources to an executable. It causes resources to be compressed. These resources will be decompressed automatically when the resource is accessed.

The -x1 option causes Resource Compiler to use the compression algorithm that is compatible with OS/2 2.0, 2.1, and 2.11, as well as OS/2 3.0 and later.

The -x2 option causes Resource Compiler to use a compression algorithm that is compatible with OS/2 3.0 and later. The -x2 option will produce smaller executable files that can access resources faster.

-x with no number defaults to -x1.

The -n option (nologo) causes Resource Compiler to suppress the display of the logo and the copyright information.

The -w2 option causes Resource Compiler to suppress the display of all warning and informational messages. Errors and fatal errors continue to be displayed.

The -h and -? options cause Resource Compiler to display a summary of the available options and environment variables that it uses. When you use these options, Resource Compiler does not read any input files. Entering "RC16" on the command line with no operands displays the same information.

Help

To display Resource Compiler help, type `RC` with no parameters, at a command prompt. The appropriate copyright statement will be displayed, along with a list of Resource Compiler options. You can also display this list by using the command-line options -h or -?.

```
Usage:  rc16 [<options>] <.RC input file> [<.EXE output file>]
   or:  rc16 [<options>] <.RES input file> [<.EXE output file>]
   or:  rc16 [<options>] -r <.RC input file> [<.RES output file>]

        -d <defname>     - Preprocessor define
        -D <defname>     - Preprocessor define
        -i <path>        - Include file path
        -r               - Create .res file
        -p               - Pack - 386 resources will not cross 64K boundaries
        -x[1|2]          - Exepack - Compress resources, using method 1 or 2
        -cc cc           - Country code
        -cp cp | lb,tb,...       - DBCS codepage or lead/trail byte info
        -n               - Don't show logo
        -w2              - Suppress warnings
        -?               - Access Help
        -h               - Access Help

Environment variables:
        DBCS=cp | lb,tb,...
        TMP=<temporary file path>
        TEMP=<temporary file path>
        INCLUDE=<include file path>; ...
```

Note: Option -X2 will compress executable files that run only on OS/2 versions 3.0 and later.

Resource Script Files

This topic describes the resource script file used to define your application resources and explains how to compile the file and add the resources to your executable file.

Use the Resource Compiler to perform the following actions:

- Create a resource script file.
- Compile the file.
- Add the file to the executable file of your application (optional).

The following sections describe the resource script file and the RC program.

Resource Script Files

A resource script file consists of one or more resource statements that define the type, identifier, and data for each resource. For example, the following multiple-line resource statement defines a menu to be used with an application:

```
  MENU 1
  BEGIN
      MENUITEM "Alpha", 101
      MENUITEM "Beta", 102
  END
```

A resource script file is a text file you can create by using an ordinary text editor. Since some resources may contain binary data that cannot be created using a text editor, many resource statements let you specify additional files to include when compiling the resource script file. For example, the following statement defines an icon and specifies the file MYICON.ICO as containing the icon data:

```
  ICON 1 myicon.ico
```

Directives

A resource script file can also contain directives. For example, the following directive includes the header file OS2.H when RC processes the resource script file:

```
  #include <os2.h>
```

Resource script files typically have the .RC file extension. .RC is the default extension; use it for all your resource script files.

Note: Although the Resource Compiler is C-like in syntax, it is not a C compiler. Use only the Resource Compiler statements.

Directives

A directive is a Resource Compiler statement that carries out a task such as including a header file, defining constants, or conditionally compiling portions of the resource script file.

Directives

elif Directive
else Directive
endif Directive
if Directive
ifdef Directive
ifndef Directive

Using the Resource Compiler

The Resource Compiler (RC) compiles a resource script file to create a new file called a binary resource file.

The binary resource file can be added to the executable file of the application, replacing any existing resources in that file.

You can start RC in any of three ways.

- Compile and add a resource definition file to an executable file
- Compile a resource script file
- Add a binary resource file to an executable file

The RC command line has the following three basic forms:

```
rc16 resource-script-file [executable-file]

rc16 resource-file [executable-file]

rc16 -r resource-script-file [resource-file]
```

Note: The third option does not add to the executable file.

The **resource-script-file** field must be the file name of the resource script file to be compiled. If the file is not in the current directory, you must provide a full path. If you provide a file name without specifying an extension, RC automatically appends the .RC extension to the name.

The **executable-file** field must be the name of the executable file to receive the compiled resources. This is a file having an extension of either .EXE or .DLL. If the file is not in the current directory, you must provide a full path. If you omit the executable-file field, RC adds the compiled resources to the executable file that has the same name as the resource script file but which has the .EXE file extension. If you specify the executable-file field but omit the extension, RC will append the .EXE extension. If this executable file does not exist, RC displays an error message.

The **-r** option directs RC to compile the resource script file without adding it to an executable file. You can use this option to prepare a binary resource file that you can add to an executable file at a later time. If you do not explicitly name a binary resource file along with the -r option, RC uses the same name as the resource script file but with the .RES extension.

The **resource-file** field must be the name of the binary resource file to be added to the executable file. If the binary resource file does not already exist, RC creates it; otherwise, RC replaces the existing file. If the file is not in the current directory, you must provide a full path. The binary resource file must have the .RES extension.

For example, to compile the resource script file EXAMPLE.RC, and add the result to the executable file EXAMPLE.EXE, use the following command:

```
rc16 example
```

You do not need to specify the .RC extension. RC creates the binary resource file EXAMPLE.RES and adds the compiled resource to the executable file EXAMPLE.EXE.

To compile the resource script file EXAMPLE.RC into a binary resource file without adding the resources to an executable file, use the following command:

```
rc16 -r example
```

The compiler creates the binary resource file EXAMPLE.RES. To create a binary resource file that has a name different from the resource script file, use the following command:

```
rc16 -r example newfile.res
```

To add the compiled resources in the binary resource file EXAMPLE.RES to an executable file, use the following command:

```
rc16 example.res
```

To specify the name of the executable file, if the name is different from the resource file, use the following command:

```
rc16 example.res newfile.exe
```

To add the compiled resources to a dynamic-link-library (.DLL) file, use the following command:

```
rc16 example.res dynalink.dll
```

DBCS Code Pages and Country Codes

In addition to -r, RC offers two other command-line options: **-cp** and **-cc**. The -cp option lets you specify a code-page identifier or DBCS lead byte information. The -cc option lets you specify a country code. The syntax is as follows:

```
-cp {codepage-id | lead-byte-start, lead-byte-end,...}
-cc country-code
```

The lead-byte-start and lead-byte-end fields give the lower and upper limits of each interval of DBCS lead bytes which you are defining for the code page. The valid range for the lower and upper limit is 128 to 255. The number values in the pair should be separated by a comma. You may specify these values instead of a codepage-id. For example:

```
 -cp 140,150
```

The codepage-id or country-code field contains a valid code page or country code. For a complete list of supported code pages and country codes, see the code page table under COUNTRYCODE in the online book *OS/2 Warp Control Program Programming Guide and* Reference.

Defining Constants

The -d option lets you define up to eight symbolic constants on the command line. The syntax is as follows:

```
-d defname[=value]
```

In the previous example, defname is a name, and value is an integer constant, or an expression. The -d option is useful for passing conditional-compilation flags to the RC preprocessor.

The following example specifies a Japanese code-page identifier and also defines two symbolic constants to be passed to the preprocessor as conditional-compilation flags.

```
rc16 -cp 932 -d DEBUG -d VERSION=2 example
```

Note: To process directives in the resource script file, RC uses the files RCPP.EXE and RCPP.ERR. Be sure that these files are in the current directory or in a directory specified by your PATH environment variable. RC creates many temporary files and writes them to the directory indicated by the TMP or TEMP environment variable. If RC cannot write these temporary files to this directory, it writes them to the current directory.

About Resource Statements

Each resource statement consists of one or more keywords, numbers, character strings, constants, or file names. You combine these to define the resource type, identifier, and data.

Keywords are words that have a special meaning to the Resource Compiler. In a statement, keywords specify the resource type, the load and memory options, and the beginning and end of nested statements. You can use the RC keywords only as specified in the statement syntax.

Keywords, except for those specifying directives, can be any combination of uppercase and lowercase letters. Note that the curly braces, { and }, are reserved characters. You can use them in place of the BEGIN and END keywords.

Numbers are integers that represent coordinates, dimensions, styles, and other numeric data. You can specify numbers in decimal, octal, or hexadecimal notation:

Decimal numbers must contain decimal digits but can start with a minus sign (-) when they represent a negative number.
Hexadecimal numbers must contain hexadecimal digits (uppercase or lowercase) and must start with the characters `0x`.
Octal numbers are similar to hexadecimal numbers, except that a lowercase letter `o` replaces the `x`.

The following example shows several numbers represented in decimal, octal, and hexadecimal notation:

DECIMAL	OCTAL	HEXADECIMAL
1	0o1	0x1
10	0o12	0xA
255	0o377	0xFF
-1	0o177777	0xFFFF
65535	0o177777	0xFFFF

Statements that create controls in dialog windows and menu items in menus require that you specify an identifier for each control or menu item. Statements that create controls also require you to specify coordinates and dimensions.

Identifiers, coordinates, and dimensions are specified using integer values. Each supports a slightly different range of values. Identifiers and coordinates can use either signed or unsigned values; dimensions only use unsigned values. Coordinates and dimensions can use unsigned values from 0 through 65535; identifiers support unsigned values from 1 through 65535. In addition, identifiers can be character strings as well as numeric values.

The ranges specific to each are listed in the following table.

	Signed Range	Unsigned Range	Strings
Identifiers	-32768 through 32767	1 through 65535	Yes
Coordinates	-32768 through 32767	0 through 65535	No
Dimensions	Not applicable	0 through 65535	No

You can also use simple expressions that evaluate to a value in the appropriate range; this enables you to, for example, specify dimensions or coordinates that are relative to those of a corresponding dialog window or menu. A resource identifiers encoded as an expression must resolve to an unsigned integer, not a string.

Character strings represent names, labels, titles, and messages. A character string consists of one or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark (") is required in a string, you must include the double quotation mark twice. The meaning of each character value (that is, the character each value represents) depends on the code page (character set) defined for the resource script file.

The Resource Compiler interprets the backslash (\) as an escape character in character strings. You can include any ASCII character in a character string by specifying either \xdd, where dd is the hexadecimal representation of an ASCII character, or \nnn, where nnn is the octal representation of an ASCII character. If a backslash is required in a string, you must include the backslash twice.

In addition, when character strings are used as resource identifiers additional rules apply:

- They must be enclosed in double quotation marks ("). If a double quotation mark is needed inside the string, it is encoded as two double quotation marks in a row.
- They cannot contain any character larger than 0x7F.
- They must be delimited by whitespace, just as an integer ID is.
- Resources whose resource ID is compiled into a 16-bit value in the binary data area, such as MENUITEM with its menu-id field, or HELPSUBITEM with its child-window-id field, cannot use character strings for IDs.
- They can contain an embedded newline character by continuing the string on the following line without closing the string. When the input file represents newlines as 0x0D+0x0A or by 0x0D+0x0D+0x0A, the string is compiled with the sequence 0x0A to represent the newline.
- Duplicate string IDs are not permitted for resources of the same type. However, the same string resource identifier can be used to identify resources of different types.

  When the Resource Compiler is compiling a script file and encounters more than one resource of the same type having the same string ID, it will generate an error message and stop compiling the file. When the Resource Compiler is binding a .RES file and encounters more than one resource of the same type with the same string ID, it will generate a warning message and ignore the second resource identifier; only the first resource having the duplicated identifier will be bound to the file.

Constants are names that have been assigned values by using the define directive. A constant can represent a number, a character string, or other data. Most resource statements in a resource script file use constants, and many use the constants defined in the OS/2 header files (for example, os2.h). For this reason, you should always use the include directive to include OS2.H in your resource script file.

File names are OS/2 file names. If the specified file is not in the current directory, you must specify the drive, directory, and file name.

Resource statements have three basic forms:

Single-line statements
Multiple-line statements
Directives

Single-line statements consist of a keyword identifying the resource type, a number or character string which specifies the resource identifier, and a file name specifying the file containing the resource data. For example, this ICON statement defines an icon resource:

```
ICON 1 myicon.ico
```

The icon resource has the icon identifier 1. The file MYICON.ICO contains the icon data. The same example, using character strings for identifiers is shown below:

```
ICON "MyIcon" myicon.ico
```

Multiple-line statements consist of a keyword identifying the resource type, a number or character string which specifies the resource identifier, and, between the BEGIN and END keywords, additional resource statements that define the resource data. For example, this MENU statement defines a menu resource:

```
MENU 1
BEGIN
    MENUITEM "Alpha", 101
    MENUITEM "Beta",  102
END
```

The menu identifier is 1. The menu contains two MENUITEM statements that define the contents of the menu.

In multiple-line statements such as DLGTEMPLATE and WINDOWTEMPLATE, RC allows any level of nested statements. For example, the DLGTEMPLATE and WINDOWTEMPLATE statements typically contain a single DIALOG or FRAME statement. These statements can contain any number of WINDOW and CONTROL statements; the WINDOW and CONTROL statements can contain additional WINDOW and CONTROL statements; and so on. The nested statements let you define controls and other child windows for the dialog boxes and windows. If a nested statement creates a child window or control, the parent and owner of the new window is the window created by the containing statement. (FRAME statements occasionally create frame controls whose parent and owner windows are not the same.)

Directives consist of the reserved character # in the first column of a line, followed by the directive keyword and any additional numbers, character strings, or file names.

Binary Resource Files

The binary resource file created by the Resource Compiler consists of one or more resource entries, each in the following form:

```
struct {
    UCHAR  fResType;
    USHORT usResType;
    union {
        struct {
            UCHAR  fResID;
            USHORT resid;
        };
        UCHAR resname[];
    };
    USHORT fsOptions;
    ULONG  cb;
    BYTE   bytes[1];
};
```

The fields in each entry have the following meanings:

fRestype	Specifies whether the resource-type identifier is a string or an integer. For OS/2, the resource type is always an integer and this field is set to 0xFF.
usResType	Specifies the resource-type identifier. This value is a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a character string. The following resource types are predefined:

RT_ACCELTABLE	Accelerator table
RT_BITMAP	Bitmap
RT_CHARTBL	Character table
RT_DIALOG	Dialog template
RT_DISPLAYINFO	Display information
RT_DLGINCLUDE	Dialog include-file name
RT_FKALONG	Long-form function-key area
RT_FKASHORT	Short-form function-key area
RT_FONT	Font
RT_FONTDIR	Font directory
RT_HELPSUBTABLE	Help subtable
RT_HELPTABLE	Help table
RT_KEYTBL	Key table
RT_MENU	Menu template
RT_MESSAGE	Error-message table
RT_POINTER	Mouse-pointer shape
RT_RCDATA	Binary data
RT_STRING	String table
RT_VKEYTBL	Virtual key table
RT_RESNAMES	String ID table

fResID	Specifies whether the resource identifier is a string or an integer. For the OS/2 operating system, this field is set to 0xFF to indicate that the resource identifier is an integer.
resid	Specifies the resource identifier. This value is an unsigned integer in the range of 1 through 65535.
resname	Specifies a string resource identifier as a sequence of characters ending with a 0x00 value.
fsOptions	Specifies the load and memory options. This value can be a combination of the following:

0x0010	MOVEABLE resource. If not given, the resource is FIXED.
0x0040	PRELOAD resource. If not given, the resource is LOADONCALL.
0x1000	DISCARDABLE resource.

cb	Specifies the size of the resource (in bytes).
bytes	Contains the resource.

Note: There is a size limitation of 65280 bytes for a binary resource file.

Statements

The following statements and directives are used by the Resource Compiler (RC):

ACCELTABLE Statement
ASSOCTABLE Statement
AUTOCHECKBOX Statement
AUTORADIOBUTTON Statement
BITMAP Statement
CHECKBOX Statement
CODEPAGE Statement
COMBOBOX Statement
CONTAINER Statement
CONTROL Statement
CTEXT Statement
CTLDATA Statement
DEFAULTICON Statement
define Directive
DEFPUSHBUTTON Statement
DIALOG Statement
DLGINCLUDE Statement
DLGTEMPLATE Statement
EDITTEXT Statement
elif Directive
else Directive
endif Directive
ENTRYFIELD Statement
FONT Statement
FRAME Statement
GROUPBOX Statement
HELPITEM Statement
HELPSUBITEM Statement
HELPSUBTABLE Statement
HELPTABLE Statement
ICON Statement (Resource)
ICON Statement (Control)
if Directive
ifdef Directive
ifndef Directive
include Directive
LISTBOX Statement
LTEXT Statement
MENU Statement
MENUITEM Statement
MESSAGETABLE Statement
MLE Statement
NOTEBOOK Statement
POINTER Statement
PRESPARAMS Statement
PUSHBUTTON Statement
RADIOBUTTON Statement
RCDATA Statement
RCINCLUDE Statement
RESOURCE Statement
RTEXT Statement
SLIDER Statement
SPINBUTTON Statement
STRINGTABLE Statement
SUBITEMSIZE Statement

ACCELTABLE Statement

Syntax:

```
ACCELTABLE acceltable-id [mem-option] [code-page]
BEGIN
key-value, command[, accelerator-options]...
    .
    .
    .
END
```

Description

The ACCELTABLE statement creates a table of accelerators for an application. An accelerator is a keystroke that gives the user a quick way to choose a command from a menu or carry out some other task. An accelerator table can be loaded when needed from the executable file by using the WinLoadAccelTable function.

You can provide any number of ACCELTABLE statements in a resource script file. Each statement must specify a unique table identifier. You can provide any number of accelerator definitions in an accelerator table; however, no two definitions in a table can specify the same key.

Each accelerator definition must specify a key value and command. The WinSetAccelTable function used in the application translates the accelerator keystroke into a WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message that has the corresponding command value. The message type depends on the accelerator-option field.

acceltable-id	Specifies the accelerator-table identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string. Each accelerator table in a resource script file must have a unique identifier.	
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one of the following:	
	Option	Meaning
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
	DISCARDABLE	System discards the resource if it is no longer needed.
code-page	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.	
key-value	Specifies the character, scan, or virtual-key code of the accelerator key. The meaning depends on the accelerator-options field. The key-value field must be a single character enclosed in double-quotation marks or an integer in the range 0 through 255. If you specify an integer, you must specify the CHAR, SCANCODE, or VIRTUALKEY accelerator option; otherwise, the default option is CHAR. Integers must be in decimal or hexadecimal notation.	
command	Specifies the command value for the corresponding WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message. This value must be a signed integer in the range -32768 through 32767, or a simple expression that evaluates to an integer in that range.	
accelerator-options	Specifies the accelerator type. This value can be a combination of the following:	
	VIRTUALKEY	Specifies that the key-value field is a virtual-key code.
	SCANCODE	Specifies that the key-value field is a keyboard scan code.
	CHAR	Specifies that the key-value field is a character code.
	SHIFT	Specifies that the user must press the Shift key and the key corresponding to the key-value field to generate the accelerator.
	CONTROL	Specifies that the user must press the Ctrl key and the key corresponding to the key-value field to generate the accelerator.

ALT	Specifies that the user must press the Alt key and the key corresponding to the key-value field to generate the accelerator.
LONEKEY	Specifies that the user needs to press only the key corresponding to the key-value field to generate the accelerator.
SYSCOMMAND	Specifies that the accelerator translates to a WM_SYSCOMMAND message. If you do not include this option, the accelerator translates to a WM_COMMAND message.
HELP	Specifies that the accelerator translates to a WM_HELP message. If you do not include this option, the accelerator translates to a WM_COMMAND message.

Note: VIRTUALKEY, SCANCODE, and CHAR are mutually exclusive. SYSCOMMAND and HELP are also mutually exclusive.

Comments

If two accelerators use the same key with different Shift, Control, or ALT options, you should specify the more restrictive accelerator first in the table. For example, you should place Shift+Enter before Enter.

If you include the OS2.H header file, you can use the following constants to specify the accelerator options:

AF_ALT	AF_CHAR	AF_CONTROL
AF_HELP	AF_LONEKEY	AF_SCANCODE
AF_SHIFT	AF_SYSCOMMAND	AF_VIRTUALKEY

To combine these constants, you must use the bitwise OR (|) operator.

Example

This example creates an accelerator table whose accelerator-table identifier is 1. The table contains two accelerators: Ctrl+S and Ctrl+G. These accelerators generate WM_COMMAND messages with values of 101 and 102, respectively, when the user presses the corresponding keys.

```
ACCELTABLE 1
BEGIN
    "S", 101, CONTROL
    "G", 102, CONTROL
END
```

ASSOCTABLE Statement

Syntax:

```
ASSOCTABLE assoctable-id [load-option][mem-option] [code-page]
BEGIN
association-name, file-match-string[, extended-attribute-flag]
   [, icon-filename]
  .
  .
  .
END
```

Description

The ASSOCTABLE statement defines a file-association table for an application. This table associates the data files that an application creates with the executable file of the application. When the user selects one of these data files from File Manager, the associated application begins executing.

A file-association table can also associate icons with the data files that an application creates. The shell uses these icons to identify the data files graphically. Because a file-association table associates icons by file type, all data files having the same file type have the same icon.

You can provide any number of ASSOCTABLE statements in a resource script file, but each statement must specify a unique assoctable-id value. The file-association tables are written not only to the resources within your executable file, but also to the .ASSOC extended attribute. However, only the last file-association table specified in the resource script file is actually written to the extended attribute.

assoctable-id	Specifies the association-table identifier. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
	DISCARDABLE	System discards the resource if it is no longer needed.
code-page	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.	
association-name	Specifies the name of the file type the application recognizes. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the name, you must include the double quotation mark twice.	
file-match-string	Specifies the file-matching string of a particular type of data file that the application creates. This field must contain zero or more characters enclosed in double quotation marks. You can only use characters that are valid in OS/2 file names and extensions and the OS/2 wildcard characters question mark (?) and asterisk (*).	
extended-attribute-flag	Specifies the extended-attribute options. This value can be a combination of the following:	
	EAF_DEFAULTOWNER	Specifies that the application containing the file-association table starts when the user selects any file matching the file-match-string field from File Manager.
	EAF_REUSEICON	Specifies that the icon defined in the previous entry of the file-association table is used as the icon for the current data-file type.
	EAF_UNCHANGEABLE	Specifies that the entry should not be edited.
icon-filename	Specifies the name of the file containing an icon. File Manager uses this icon to represent all application-created data files matching the file-match-string field. The file must be in the current directory.	

AUTOCHECKBOX Statement

Syntax:

```
AUTOCHECKBOX text, id, x, y, width, height[, style]
```

The AUTOCHECKBOX statement creates an automatic-check-box control. The control is a small rectangle (check box) that contains an X when the user selects it. The specified text is displayed to the right of the check box. An X appears in the square when the user first selects the control and disappears the next time the user selects it. The AUTOCHECKBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify the style, the default style is BS_AUTOCHECKBOX and WS_TABSTOP.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in

	the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an automatic-check-box control that is labeled "Italic."

```
AUTOCHECKBOX "Italic", 101, 10, 10, 100, 100
```

AUTORADIOBUTTON Statement

Syntax:

```
AUTORADIOBUTTON text, id, x, y, width, height[, style]
```

The AUTORADIOBUTTON statement creates an automatic-radio-button control. This control is a small circle with the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control also removes the selection from any other automatic-radio-button controls in the same group. When the user selects the button again, the control removes the highlight before sending a message. The AUTORADIOBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_AUTORADIOBUTTON.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an automatic-radio-button control that is labeled "Italic."

```
AUTORADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

BITMAP Statement

Syntax:

```
BITMAP bitmap-id [load-option] [mem-option] [codepage] filename
```

The BITMAP statement defines a bit map resource for an application. A bit map resource, typically created using the Icon Editor, is a custom bit map that an application uses in its display or as an item in a menu. The BITMAP statement copies the bit-map resource from the file specified in the filename field and adds it to the application's other resources. A bit-map resource can be loaded from the executable file when needed by using the GpiLoadBitmap function.

You can provide any number of BITMAP statements in a resource script file, but each statement must specify a unique bitmap-id value.

bitmap-id	Specifies the bit-map-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the GpiLoadBitmap function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
	DISCARDABLE	System discards the resource if it is no longer needed.
codepage	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.	
filename	Specifies the name of the file containing the icon resource. If the file is not in the current directory, filename must be preceded by a full path.	

Example

This example defines a bit map whose bit-map identifier is 12. The bit-map resource is copied from the file CUSTOM.BMP.

```
BITMAP 12 custom.bmp
```

CHECKBOX Statement

Syntax:

```
CHECKBOX text, id, x, y, width, height[, style]
```

The CHECKBOX statement creates a check-box control. The control is a small rectangle (check box) that has the specified text displayed to the right. The control highlights the rectangle and sends a message to its parent window when the user selects the control. The CHECKBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_CHECKBOX and WS_TABSTOP.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a check-box control that is labeled "Italic."

```
CHECKBOX "Italic", 101, 10, 10, 100, 100
```

CODEPAGE Statement

Syntax:

```
CODEPAGE codepage-id
```

The CODEPAGE statement sets the code page for all subsequent resources. The code page specifies the character set used for characters in the resource.

If the CODEPAGE statement is not given in a resource script file, RC uses the code page set up for the individual system. If more than one CODEPAGE statement is given in the file, each CODEPAGE statement applies to the resource statements between it and the next CODEPAGE statement.

codepage-id	Identifies the code page to be used for subsequent resources. For a complete list of supported code pages, refer to the "COUNTRYCODE" section of the *Control Program Programming Guide and Reference* .

Comments

You may also specify a code page by placing a code-page identifier in the load-options or memory-options field of any RC statement that uses those fields.

Example

In this example, the code page for the character-string resources is set to Portuguese (860).

```
CODEPAGE 860

STRINGTABLE
BEGIN
    1 "Filename not found"
    2 "Cannot open file for reading"
END
```

COMBOBOX Statement

Syntax:

```
  COMBOBOX text, id, x, y, width, height[, style]
```

The COMBOBOX statement creates a combination-box control. This control combines a list-box control with an entry-field control. It allows the user to place the selected item from a list box into an entry field.

The COMBOBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_COMBOBOX. If you do not specify a style, the default style is CBS_SIMPLE, WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

text	Specifies text that is displayed in the entry field of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_COMBOBOX. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a combination-box control.

```
  COMBOBOX "", 101, 10, 10, 24, 50
```

CONTAINER Statement

Syntax:

```
  CONTAINER  id, x, y, width, height [,style]
```

The CONTAINER statement creates a container control within a dialog window. The container control is a visual component that holds objects. The CONTAINER statement defines the identifier, position, dimensions, and attributes of a container control. The predefined class for this control is WC_CONTAINER. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and CCS_SINGLESEL.

id	Specifies the control identifier. This value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window containing the container control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window containing the container control.
width	Specifies the width of the control. This value is any integer 0 through 65535, or an expression consisting of integers and

	the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_CONTAINER. Use the bitwise OR (|) operator to combine styles.

Comments

A CONTAINER statement is only used in a DIALOG or WINDOW statement.

Example

This example creates a container control at position (30,30) within the dialog window. The container has a width of 70 character units and a height of 25 character units. Its resource identifier is 301. The default style CCS_SINGLESEL has been overridden by the style specification CCS_MULTIPLESEL. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define IDC_CONTAINER    301
#define IDD_CONTAINERDLG 504
DIALOG "Container", IDD_CONTAINERDLG, 23, 6, 120, 280, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
     CONTAINER   IDC_CONTAINER, 30, 30, 70, 200, CCS_MULTIPLESEL |
                       WS_GROUP
  END
```

CONTROL Statement

Syntax:

```
  CONTROL text, id, x, y, width, height, class[, style]
          [data-definitions]
  [ BEGIN
  control-definition
      .
      .
      .
  END ]
```

The CONTROL statement defines a control as belonging to the specified class. The statement defines the position and dimensions of the control within the parent window, as well as the control style. The CONTROL statement is most often used in a DIALOG or WINDOW statement.

Typically, several CONTROL statements are used in each DIALOG statement, and each CONTROL statement must have a unique identifier value. The optional BEGIN and END statements enclose any CONTROL statements that may be given with the control. CONTROL statements given in this manner represent child windows belonging to the control created by the CONTROL statement.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. In the appropriate styles, a tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the parent window.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the parent window.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
class	Specifies the control class. This value can be one of the control classes specified in the "Control Classes" table, in the Presentation Manager Programming Reference, or the name of the control class, enclosed in double quotation marks.

style	Specifies the control style. This value can be a combination of control styles. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the control. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the control.

Comments

The CONTROL statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. But typically, a CONTROL statement contains no such statements.

Example

This example creates a pushbutton control with the WS_TABSTOP and WS_VISIBLE styles.

```
  CONTROL "OK", 101, 10, 10, 20, 50, WC_BUTTON, BS_PUSHBUTTON |
                                              WS_TABSTOP    |
                                              WS_VISIBLE
```

CTEXT Statement

Syntax:

```
  CTEXT text, id, x, y, width, height[, style]
```

The CTEXT statement creates a centered-text control. The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The CTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_CENTER, and WS_GROUP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a centered-text control that is labeled "Filename."

```
  CTEXT "Filename", 101, 10, 10, 100, 100
```

CTLDATA Statement

Syntax:

```
CTLDATA word-value[, word-value][...]

CTLDATA string

CTLDATA MENU
BEGIN
menuitem-definition
   .
   .
   .
END
```

The CTLDATA statement defines control data for a custom dialog box, window, or control. The statement has three basic forms to permit specifying a menu or specifying data in words or characters. The data can be in any format, since only your window procedure will use it. The window procedure of the dialog box, window, or control receives this data when the item is created. It is up to the window procedure to process the data.

word-value	Specifies a 16-bit value. This value must be a signed integer in the range -32768 through 32767 or an expression that evaluates in that range.
string	Specifies a string of 8-bit characters. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice.
menuitem-definition	Specifies a MENUITEM or SUBMENU statement. These statements define the individual commands or submenus in the given menu. For details about these statements, see MENUITEM Statement and SUBMENU Statement.

Comments

CTLDATA is often used to supply data that controls the subsequent operation of the custom window. For example, the CTLDATA statement may contain extended style bits - that is, style bits designed specifically for your customized window.

You should reserve the CTLDATA statement for window classes that you create yourself.

Example

This example creates a menu for the window created with the WINDOW statement.

```
WINDOWTEMPLATE 1
BEGIN
    WINDOW "Sample", 1, 0, 0, 100, 100, "MYCLASS", 0, FCF_STANDARD
    CTLDATA MENU
    BEGIN
        MENUITEM "Exit", 101
    END
END
```

DEFAULTICON Statement

Syntax:

```
DEFAULTICON filename.ico
```

This statement installs the named icon file definition under the ICON Extended Attribute of the program file. An icon with an icon-id of 1 is the default unless you supply a different icon.

Example

```
DEFAULTICON myicon.ico
```

----

define Directive

Syntax:

```
  define name value
```

The define directive assigns the given value to the specified name. All subsequent occurrences of the name are replaced by the value.

name	Specifies the name to be defined. This name can be any combination of letters, digits, or underscore characters which does not begin with a digit.
value	Specifies any integer, character string, or line of text. This value can contain another defined name, which creates a level of nested defines. You are limited to 64 levels of nested defines.

Example

This example assigns values to the names "NONZERO" and "USERCLASS".

```
  #define     NONZERO     1
  #define     USERCLASS   "MyControlClass"
```

DEFPUSHBUTTON Statement

Syntax:

```
  DEFPUSHBUTTON text, id, x, y, width, height[, style]
```

The DEFPUSHBUTTON statement creates a default pushbutton control. The control is a round-cornered rectangle containing the given text. The rectangle has a bold outline to represent that it is the default response for the user. The control sends a message to its parent window when the user chooses the control. The DEFPUSHBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON, BS_DEFAULT, and WS_TABSTOP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a default pushbutton control that is labeled "Cancel."

```
DEFPUSHBUTTON "Cancel", 101, 10, 10, 24, 50
```

DIALOG Statement

Syntax:

```
  DIALOG text, id, x, y, width, height[, style[, framectl]]
         [data-definitions]
  BEGIN
  control-definition
      .
      .
      .
  END
```

The DIALOG statement defines a window that an application can use to create dialog boxes. The statement defines the position and dimensions of the dialog box on the screen, as well as the dialog-box style. The DIALOG statement is most often used in a DLGTEMPLATE statement.

Typically, you use only one DIALOG statement in each DLGTEMPLATE statement, and the DIALOG statement contains at least one control definition.

text	Specifies the dialog-box title if the style specifies a title bar. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the title, you must include the double quotation mark twice.
id	Specifies the dialog-box identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the dialog box. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units, but its exact meaning depends on the dialog style. See the "Comments" section for details.
y	Specifies the y-coordinate of the lower-left corner of the dialog box. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units, but its exact meaning depends on the dialog style. See the "Comments" section for details.
width	Specifies the width of the dialog box. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the dialog box. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
style	Specifies the dialog-box style. This value can be any of the window, dialog-box, or frame styles. You can use the bitwise OR (|) operator to combine styles.
framectl	Specifies the styles for frame controls belonging to the dialog box. This value can be any of the frame-control styles specified in the "Frame-Control Flags" table in the Presentation Manager Programming Reference. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the dialog box. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the dialog box.

Comments

The exact meaning of the coordinates depends on the style defined by the style field. For dialog boxes with FS_SCREENALIGN style, the coordinates are relative to the origin of the display screen. For dialog boxes with the style FS_MOUSEALIGN, the coordinates are relative to the position of the mouse pointer at the time the dialog box is created. For all other dialog boxes, the coordinates are relative to the origin of the parent window.

The DIALOG statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a DIALOG statement contains one or more CONTROL statements.

Example

This example creates a dialog box that is labeled "Disk Error."

```
DLGTEMPLATE 1
BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

DLGINCLUDE Statement

Syntax:

```
  DLGINCLUDE id filename
```

The DLGINCLUDE statement adds the specified file to the resource file. The DLGINCLUDE statement is typically used to let the application access the definitions file for the dialog box with the corresponding identifier. The file named by filename must contain the define directives used by the dialog box.

You can provide any number of DLGINCLUDE statements in a resource script file, but each must have a unique identifier.

id	Specifies the dialog-box identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
filename	Specifies the name of the file containing the define directives. If the file is not in the current directory, filename must be preceded by a full path.

Example

This example includes the name of the definition file dlgdef.h. The dialog-box identifier is 5.

```
DLGINCLUDE 5 \\INCLUDE\\DLGREF.H
```

DLGTEMPLATE Statement

Syntax:

```
  DLGTEMPLATE dialog-id  [load-option] [mem-option] [codepage]
  BEGIN
  dialog-definition
      .
      .
      .
  END
```

The DLGTEMPLATE statement creates a dialog-box template. A dialog-box template consists of a series of statements that define the identifier, load and memory options, dialog-box dimensions, and controls in the dialog box. The dialog-box template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of dialog-box templates in a resource script file, but each template must have a unique dialog-id value.

dialog-id	Specifies the dialog-box identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:
	PRELOAD — System loads the resource when the application starts.
	LOADONCALL — System loads the resource when the application calls the WinLoadDlg function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:
	FIXED — System keeps the resource at a fixed memory location.
	MOVEABLE — System moves the resource as necessary to compact memory.
	DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code-page value. For a list of valid code pages see CODEPAGE Statement.
dialog-definition	Specifies a DIALOG statement. The statement defines the dimensions and style of the given dialog box. For details about the statement, see DIALOG Statement.

Comments

A DLGTEMPLATE statement can actually contain DIALOG, CONTROL, and WINDOW statements. Typically, you include only one DIALOG statement.

Example

This example uses a DLGTEMPLATE statement to create a dialog box.

```
DLGTEMPLATE ID_GETTIMER
BEGIN
    DIALOG "Timer", 1, 10, 10, 100, 40
    BEGIN
        LTEXT "Time (0 - 15):", 4, 8, 24, 72, 12
        ENTRYFIELD "0", ID_TIME, 80, 28, 16, 8, ES_MARGIN
        DEFPUSHBUTTON "Enter", ID_TIMEOK, 10, 6, 36, 12
        PUSHBUTTON "Cancel", ID_TIMECANCEL, 52, 6, 40, 12
    END
END
```

EDITTEXT Statement

Syntax:

```
  EDITTEXT text, id, x, y, width, height [,style]
```

The EDITTEXT statement creates an entry-field control. This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters.

The EDITTEXT statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in

	dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_ENTRYFIELD. You can use the bitwise OR (|) operator to combine styles.

Comments

The EDITTEXT control statement is identical to the ENTRYFIELD control statement.

Use the EDITTEXT statement only in a DIALOG or WINDOW statement.

Example

This example creates an entry-field control that is not labeled.

```
   EDITTEXT "", 101, 10, 10, 24, 50
```

----

elif Directive

Syntax:

```
  elif constant-expression
```

The elif directive marks an optional clause of a conditional-compilation block defined by a ifdef, ifndef, or if directive. The directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, elif directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after endif. If the constant expression is zero, elif directs the compiler to skip to the next endif, else, or elif directive. You can use any number of elif directives in a conditional block.

constant-expression	Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

In this example, elif directs the compiler to process the second BITMAP statement only if the value assigned to the name "Version" is less than 7. The elif directive itself is processed only if Version is greater than or equal to 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#elif Version < 7
BITMAP 1 userbox.bmp
#endif
```

----

else Directive

Syntax:

```
  else
```

The else directive marks an optional clause of a conditional-compilation block defined by a ifdef, ifndef, or if directive. The else directive must be the last directive before the endif directive.

This directive has no arguments.

Example

This example compiles the second BITMAP statement only if the name "DEBUG" is not defined.

```
    #ifdef DEBUG
        BITMAP 1 errbox.bmp
    #else
        BITMAP 1 userbox.bmp
    #endif
```

----

endif Directive

Syntax:

```
  endif
```

The endif directive marks the end of a conditional-compilation block defined by a ifdef directive. One endif is required for each if, ifdef, or ifndef directive.

This directive has no arguments.

ENTRYFIELD Statement

Syntax:

```
  ENTRYFIELD text, id, x, y, width, height , [style]
```

The ENTRYFIELD statement creates an entry-field control. This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters. The ENTRYFIELD statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_ENTRYFIELD. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an entry-field control that is not labeled.

```
ENTRYFIELD "", 101, 10, 10, 24, 50
```

FONT Statement

Syntax:

```
  FONT font-id  [load-option] [mem-option] [codepage]  filename
```

The FONT statement defines a font resource for an application. A font resource, typically created by using the OS/2 Font Editor, is a bit map defining the shape of the individual characters in a character set. The FONT statement copies the font resource from the file specified in the filename field and adds it to the other resources of the application. A font resource can be loaded from the executable file when needed by using the GpiLoadFonts function.

You can provide any number of FONT statements in a resource script file, but each statement must specify a unique font-id value.

font-id	Specifies the font-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the GpiLoadFonts function. This is the default option.
mem-option	Specifies how the system manages the resource when it is i memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and
	DISCARDABLE.	
codepage	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.	
filename	Specifies the name of the file containing the font resource. If the file is not in the current directory, filename must be preceded by a full path.	

Example

This example defines a font whose font identifier is 5. The font resource is copied from the file cmroman.fon.

```
FONT 5 cmroman.fon
```

FRAME Statement

Syntax:

```
  FRAME text, id, x, y, width, height[, style[, framectl]]
        data-definitions
  [ BEGIN
  window-definition
      .
```

```
        .
        .
    END ]
```

The FRAME statement defines a frame window. The statement defines the title, identifier, position, and dimensions of the frame window, as well as the window style. The FRAME statement is most often used in a WINDOWTEMPLATE statement, and typically, only one FRAME statement is used. The FRAME statement, in turn, typically contains at least one WINDOW statement that defines the client window belonging to the frame window.

The frame window has no default style. You must use the **framectl** field to define additional frame controls, such as a title bar and system menu, to be created when the frame window is created. If the text field is not empty, the statement automatically adds a title-bar control to the frame window, whether or not you specify the FCF_TITLEBAR style. Frame controls are given default styles and control identifiers depending on their class. For example, a title-bar control receives the identifier FID_TITLEBAR.

text	Specifies the title of the frame window. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the name, you must include the double quotation mark twice.
id	Specifies the window identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified window.
y	Specifies the y-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified window.
width	Specifies the width of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the window. This value must be a integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the frame and window styles. This value can be a combination of frame styles. You can use the bitwise OR (|) operator to combine styles.
framectl	Specifies the styles of frame controls belonging to the frame window. This value can be a combination of the styles specified in the "Frame-Control Styles" table in the Presentation Manager Programmers Reference. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the frame window. For more information, see CTLDATA Statement and PRESPARAMS Statement.
window-definition	Specifies a WINDOW statement or any one of several predefined control statements. These statements define the style, position, and dimensions of windows or controls in the frame window.

Comments

The FRAME statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a FRAME statement contains one WINDOW statement.

Example

This example creates a standard frame window, with a title bar, a system menu, minimize and maximize boxes, and a vertical scroll bar. The FRAME statement contains a WINDOW statement defining the client window belonging to the frame window.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, 0,
            FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
END
```

GROUPBOX Statement

Syntax:

```
  GROUPBOX text, id, x, y, width, height [, style]
```

The GROUPBOX statement creates a group-box control. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner. The GROUPBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_GROUPBOX and WS_TABSTOP.

text	Specifies text that appears in the upper-left corner of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a group-box control that is labeled "Options."

```
GROUPBOX "Options", 101, 10, 10, 100, 100
```

HELPITEM Statement

Syntax:

```
  HELPITEM application-window-id, help-subtable-id, extended-helppanel-id
```

The HELPITEM statement defines the help items in a help table. The statement, permitted only in a HELPTABLE statement, specifies the resource identifier of an application window for which help is provided, and the resource identifiers of the help subtable and extended help panel associated with the application window.

You can provide any number of HELPITEM statements in a HELPTABLE statement. You should provide one HELPITEM statement for each application window for which help is provided.

application-window-id	Specifies the resource identifier of an application window for which help is provided. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
help-subtable-id	Specifies the resource identifier of the help subtable associated with the specified application window. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
extended-helppanel-id	Specifies the resource identifier of the extended help panel associated with the specified application window. This value must be an integer in the range 0 through 65535. However, in IPF a panel-id must be an integer in the range of 0 to 64000.

Example

This example defines a help item that associates a help subtable called IDSUB_FILEMENU and an extended help panel called

IDEXT_APPHLP with an application window called IDWIN_FILEMENU.

```
HELPITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
```

HELPSUBITEM Statement

Syntax:

```
  HELPSUBITEM child-window-id, helppanel-id [ , integer...]
```

The HELPSUBITEM statement defines the help subitems in a help subtable. This statement, which is permitted only in a HELPSUBTABLE statement, specifies the identifier of a child window for which help is provided, the identifier of the help panel associated with the child window, and one or more optional, application-defined integers.

You can provide any number of HELPSUBITEM statements in a HELPSUBTABLE statement. You should provide one HELPSUBITEM statement for each child window for which help is provided.

child-window-id	Specifies the resource identifier of the child window for which help is provided. Character strings cannot be used as resource identifiers for this statement.
helppanel-id	Specifies the resource identifier of the help panel associated with the specified child window.
integer	Specifies optional, application-defined integers. If you use this field, you must include the SUBITEMSIZE statement in the help subtable to specify the size, in words, of each help subitem in the help subtable. For details about this statement, see SUBITEMSIZE Statement.

Example

This example defines a help subitem that associates a child window called IDCLD_FILEMENU with a help panel called IDHP_FILEMENU.

```
HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU
```

HELPSUBTABLE Statement

Syntax:

```
  HELPSUBTABLE helpsubtable-id
    SUBITEMSIZE size
  BEGIN
  helpsubitem-definition
       .
       .
       .
  END
```

The HELPSUBTABLE statement defines the contents of a help-subtable resource. A help-subtable resource contains a help-subitem entry for each item that can be selected in an application window. Each of these items should be a child window of the application window specified in the help-table resource. The help subtable should contain a help subitem for each control, child window, and menu item in the application window.

You can provide any number of HELPSUBTABLE statements in a resource script file, but each statement must specify a unique helpsubtable-id value. You can also provide any number of helpsubitem-definition statements in the help subtable. These specify the child window for which help is provided, the help panel containing the help text for the child window, and one or more application-defined integers.

If you include optional integers in the helpsubitem-definition statements, you must also include a SUBITEMSIZE statement to specify the size, in words, of each help subitem. All help subitems in a help subtable must be the same size. The default size is two words per help subitem. (No SUBITEMSIZE statement is needed if you do not include optional integers in the helpsubitem-definition statement.)

helpsubtable-id

Specifies the resource identifier of the help subtable. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.

helpsubitem-definition

Specifies a HELPSUBITEM statement. A HELPSUBITEM statement specifies a child window, the help panel associated with the child window, and one or more optional, application-defined integers. For details about this statement, see HELPSUBITEM Statement.

Example

This example creates a help-subtable resource whose help-subtable identifier is IDSUB_FILEMENU. Each HELPSUBITEM statement specifies a child window and a help panel.

```
HELPSUBTABLE IDSUB_FILEMENU
BEGIN
    HELPSUBITEM IDCLD_OPEN, IDPNL_OPEN
    HELPSUBITEM IDCLD_SAVE, IDPNL_SAVE
END
```

HELPTABLE Statement

Syntax:

```
  HELPTABLE helptable-id
  BEGIN
  helpitem-definition
      .
      .
      .
  END
```

The HELPTABLE statement defines the contents of a help-table resource. A help-table resource contains a help-item entry for each application window, dialog box, and message box for which help is provided.

You can provide any number of HELPTABLE statements in a resource script file, but each statement must specify a unique helptable-id value. You can also provide any number of helpitem-definition statements in the help table. These specify the application windows for which help is provided, the help subtables associated with each application window, and the extended help panels associated with each application window.

helptable-id	Specifies the resource identifier of the help table. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.
helpitem-definition	Specifies a HELPITEM statement. A HELPITEM statement specifies an application window and the associated help subtable and extended help panel. For details about this statement, see HELPITEM Statement.

Example

This example creates a help-table resource whose help-table identifier is 1. Each HELPITEM statement specifies an application window, a help subtable, and an extended help panel.

```
HELPTABLE 1
BEGIN
    HELPITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
    HELPITEM IDWIN_EDITMENU, IDSUB_EDITMENU, IDEXT_APPHLP
END
```

ICON Statement (Resource)

Syntax:

```
  ICON icon-id  [load-option] [ mem-option] [codepage] filename
```

This form of the ICON statement defines an icon resource for an application. An icon resource, typically created by using Icon Editor, is a bit map defining the shape of the icon to be used for a given application. The ICON statement copies the icon resource from the file specified in the filename field and adds it to the application's other resources. An icon resource can be loaded when creating a window by using the WinCreateStdWindow function with the FS_ICON style.

You can provide any number of ICON statements in a resource script file, but each statement must specify a unique icon-id value.

icon-id	Specifies the icon-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string. An icon-id of 1 has a special meaning; for details, see the "Comment" section.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinCreateStdWindow function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and
	DISCARDABLE.	
codepage	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.	
filename	Specifies the name of the file containing the icon resource. If the file is not in the current directory, filename must be preceded by a full path.	

Comments

An icon with an icon-id of 1 is the default icon. The RC program writes the icon not only to the resources in your executable file, but also as the .ICON extended attribute. File Manager will display this icon next to the name of the executable file.

Example

This example defines an icon whose icon identifier is 11. The icon resource is copied from the file custom.ico.

```
ICON 11 custom.ico
```

ICON Statement (Control)

Syntax:

```
  ICON icon-id, id, x, y, width, height, [style]
```

This form of the ICON statement creates an icon control. This control is an icon displayed in a dialog box. The ICON statement, which you can use only in a DIALOG or WINDOW statement, defines the icon-resource identifier, icon-control identifier, position, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_ICON. For the ICON statement, the width and height fields are ignored; the icon automatically sizes itself.

icon-id	Specifies the resource identifier of an icon that is defined elsewhere in the resource file.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.

y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies a reserved value. Can be set to zero.
height	Specifies a reserved value. Can be set to zero.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an icon control whose icon identifier is 99.

```
ICON 99, 101, 10, 10, 0, 0
```

----

if Directive

Syntax:

```
  if constant-expression
```

The if directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, if directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after the endif directive. If the constant expression is zero, if directs the compiler to skip to the next endif, else, or elif directive.

constant-expression	Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

This example compiles the BITMAP statement only if the value assigned to the name "Version" is less than 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#endif
```

----

ifdef Directive

Syntax:

```
  ifdef name
```

The ifdef directive controls conditional compilation of the resource file by checking the specified name. If the name has been defined by using a define directive or by using the -d command-line option of rc, ifdef directs the compiler to continue with the statement immediately after the ifdef directive. If the name has not been defined, ifdef directs the compiler to skip all statements up to the next endif directive.

name	Specifies the name to be checked by the directive.

Example

This example compiles the BITMAP statement only if the name "Debug" is defined.

```
#ifdef Debug
BITMAP 1 errbox.bmp
#endif
```

----

ifndef Directive

Syntax:

```
  ifndef name
```

The ifndef directive controls conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed by using the undef directive, ifndef directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after the endif directive. If the name is defined, ifndef directs the compiler to skip to the next endif, else, or elif directive.

name	Specifies the name to be checked by the directive.

Example

This example compiles the BITMAP statement only if the name "Optimize" is not defined.

```
#ifndef Optimize
BITMAP 1 errbox.bmp
#endif
```

----

include Directive

Syntax:

```
  include filename
```

The include directive causes RC to process the file specified in the filename field. This file should be a header file that defines the constants used in the resource script file. Only the #define directives in the specified file are processed; all other statements are ignored by the Resource Compiler.

filename	Specifies the OS/2 name of the file to be included. This value must be an ASCII string enclosed either in double quotation marks (if the file is in the current directory) or in less-than and greater-than characters (<>) (if the file is in the directory specified by -i command-line options or by the INCLUDE environment variable). You must give a full path enclosed in double quotation marks if the file is not in the current directory or in the directory specified by -i command-line options or by the INCLUDE environment variable.

Comments

The filename field is handled as a C string. Therefore, you must include two backslashes wherever one is required in the path. (As an alternative, you can use a single forward slash (/) instead of two backslashes.)

Example

This example processes the header files OS2.H and HEADERS\MYDEFS.H\I while compiling the resource script file.

```
#include <os2.h>
#include "headers\\\\mydefs.h"
```

LISTBOX Statement

Syntax:

```
  LISTBOX id, x, y, width, height[, style]
```

The LISTBOX statement creates commonly used controls for a dialog box or window. The control is a rectangle containing a list of user-selectable strings, such as file names.

The LISTBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_LISTBOX. If you do not specify a style, the default style is WS_TABSTOP.

id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_LISTBOX. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a list-box control whose identifier is 101.

```
LISTBOX 101, 10, 10, 100, 100
```

LTEXT Statement

Syntax:

```
  LTEXT text, id, x, y, width, height [, style]
```

The LTEXT statement creates a left-aligned text control. The control is a simple rectangle displaying the given text left-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The LTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_LEFT, and WS_GROUP.

text	Specifies text that is left-aligned in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.

style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a left-aligned text control that is labeled "Filename."

```
LTEXT "Filename", 101, 10, 10, 100, 100
```

MENU Statement

Syntax:

```
  MENU menu-id  [load-option] [mem-option] [codepage]
  BEGIN
  menuitem-definition
       .
       .
       .
  END
```

The MENU statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user choose commands from a list of command names. A menu resource can be loaded from the executable file when needed by using the WinLoadMenu function.

You can provide any number of MENU statements in a resource script file, but each statement must specify a unique menu-id value. You can provide any number of menuitem-definition statements in the menu. These define the submenus and menu items (commands) in the menu. The order of the statements defines the order of the menu items.

menu-id	Specifies the menu-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinLoadMenu function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is
	MOVEABLE and DISCARDABLE.	
codepage	Specifies a codepage value. For a list of valid code pages see CODEPAGE Statement.	
menuitem-definition	Specifies a PRESPARAMS, MENUITEM, or SUBMENU statement. You can use one or more PRESPARAMS statements to control the appearance of a menu, such as the font and the foreground and background colors. If used, PRESPARAMS statements must be the first statements following the BEGIN keyword. For details about the PRESPARAMS statement, see PRESPARAMS Statement. MENUITEM and SUBMENU statements define the individual commands or submenus in the given menu. For details about these statements, see MENUITEM Statement and SUBMENU Statement.	

Example

This example creates a menu resource whose menu identifier is 1. The menu contains a menu item named Alpha and a submenu named Beta. The submenu contains two menu items, Item 1 and Item 2.

```
MENU 1
```

```
BEGIN
    MENUITEM "Alpha", 100
    SUBMENU "Beta", 101
    BEGIN
        MENUITEM "Item 1", 200
        MENUITEM "Item 2", 201, , MIA_CHECKED
    END
END
```

MENUITEM Statement

Syntax:

```
  MENUITEM text, menu-id[,  menuitem-style] [, menuitem-attribute]
```

The MENUITEM statement creates a menu item for a menu. The statement, permitted only in a MENU or SUBMENU statement, defines the text, identifier, and attributes of a menu item. The system displays the text when it displays the corresponding menu. If the user chooses the menu item, the system generates a WM_COMMAND message that includes the specified menu-item identifier and sends it to the window owning the menu.

MENUITEM SEPARATOR

The alternative form of the MENUITEM statement, MENUITEM SEPARATOR, creates a menu separator. A menu separator is a horizontal dividing bar between two menu items in a submenu. The separator is not active - that is, the user cannot choose it, it has no text associated with it, and it has no identifier.

text	Specifies the text of the menu item. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice. The tilde character (~) and the \t and \a character combinations have special meanings in the string; for details, see the "Comments" section. If the menuitem-style field is MIS_BITMAP, item-name must be a bit-map identifier instead of a name. The bit-map identifier must have been previously defined using a BITMAP statement, must be preceded by the \b character, and must be enclosed in double quotation marks.
menu-id	Specifies the menu-item identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Duplicate menu-item identifiers are allowed, but menu items with non-unique identifiers cannot receive messages. Character strings cannot be used as resource identifiers for this statement.
menuitem-style	Specifies the menu-item style. This value can be a combination of the following:

MIS_BITMAP	Specifies that item-name is a bit map identifier.
MIS_BREAK	Specifies that the menu has multiple columns of items in one pull-down menu or multiple lines of menus in the top-level menu.
MIS_BREAKSEPARATOR	Specifies that the menu has a vertical line between the columns in a pull-down menu.
MIS_BUTTONSEPARATOR	Specifies that the user can activate the menu item only by using the mouse. The text is centered in the item, rather than left justified. This option is used for the Help item on the right side of the menu bar.
MIS_HELP	Specifies that the menu item generates a WM_HELP message.
MIS_OWNERDRAW	Specifies that the menu item is drawn by

		the owner window.
	MIS_SEPARATOR	Specifies that the menu item is a menu separator.
	MIS_STATIC	Specifies that the user cannot choose the menu item.
	MIS_SUBMENU	Specifies that the MENUITEM statement is to be treated as a SUBMENU statement. When you specify this option, you must follow the MENUITEM statement with a BEGIN and END clause, as in a SUBMENU statement. You may include a PRESPARAMS statement immediately after the BEGIN keyword.
	MIS_SYSCOMMAND	Specifies that the menu item generates a WM_SYSCOMMAND message.
	MIS_TEXT	Specifies that item-name is a character string. This is the default option.
menuitem-attribute	Specifies the menu-item attributes. This value can be a combination of the following:	
	MIA_CHECKED	Places a check mark next to the menu-item name.
	MIA_DISABLED	Disables the menu item, preventing the system from generating a message when the user chooses the command.
	MIA_FRAMED	Places a frame (heavy border) around the menu item.
	MIA_HILITED	Places a highlight on the menu-item name when it is displayed, by inverting the name and background.
	MIA_NODISMISS	Causes a submenu or menu item to remain displayed after the user chooses an item.

Comments

You can use the \t or \a character combination in any item name. The \t character inserts a tab when the name is displayed and is typically used to separate the menu-item name from the name of an accelerator key. The \a character aligns to the right all text that follows it. These characters are intended to be used for menu items in submenus only. The width of the displayed submenu is always adjusted so that there is at least one space (and usually more) between any pieces of text separated by a \t or a \a. (When compiling the menu resource, the compiler stores the \t and \a characters as control characters. For example, the \t is stored as 0x09.)

A tilde (~) character in the item name indicates that the following character is used as a mnemonic character for the item. When the menu is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the menu item by pressing the key corresponding to the underlined mnemonic character.

Example

This example creates a menu item named Alpha. The item identifier is 101.

```
MENUITEM "Alpha", 101
```

This example creates a menu item named Beta. The item identifier is 102. The menu item has a text style and a checked attribute.

```
MENUITEM "Beta", 102, MIS_TEXT, MIA_CHECKED
```

This example creates a menu separator between menu items named Gamma and Delta.

```
MENUITEM "Gamma", 103
MENUITEM SEPARATOR
MENUITEM "Delta", 104
```

This example creates a menu item that has a bit map instead of a name. The bit-map identifier, 1, is first defined using a BITMAP statement. The identifier for the menu item is 301. Note that a sign must be placed in front of the bit map identifier in the MENUITEM statement.

```
BITMAP 1 mybitmap.bmp

MENUITEM "#1", 301, MIS_BITMAP
```

MESSAGETABLE Statement

Syntax:

```
MESSAGETABLE  [load-option] [mem-option] [codepage]
BEGIN
string-id string-definition
    .
    .
    .
END
```

The MESSAGETABLE statement creates one or more string resources for an application. A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using the DosGetResource function with the RT_MESSAGE resource type. RT_MESSAGE resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential identifier. The resource string identifier is not necessarily the same as the identifier specified when using DosGetResource. The formula for calculating the identifier of the resource bundle, for use in DosGetResource, is as follows:

```
bundle ID = (id / 16) + 1
```

where id is the string identifier assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated identifier), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of MESSAGETABLE statements in a resource script file. The compiler treats all the strings from the various MESSAGETABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

Although the MESSAGETABLE and STRINGTABLE statements are nearly identical, most applications use the STRINGTABLE statement instead of the MESSAGETABLE statement to create string resources.

load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.
string-id	Specifies the character-string identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. The value can be specified in decimal or hexadecimal notation. Each string identifier in a resource script file must be unique.
string-definition	Specifies a character string. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must provide the double quotation mark twice.

Comments

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
MESSAGETABLE
BEGIN
    1 "Filename not found"
    2 "Cannot open file for reading"
END
```

MLE Statement

Syntax:

```
  MLE text, id, x, y, width, height[, style]
```

The MLE statement creates a multiple-line entry-field control. The control is a rectangle in which the user can type and edit multiple lines of text. The control displays a pointer when the user selects it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters. The MLE statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_MLE. If you do not specify a style, the default style is MLS_BORDER, WS_GROUP, and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. If the MLS_READONLY style is not specified, the user can edit the text. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_MLE. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a multiple-line entry-field control that is not labeled.

```
MLE "", 101, 10, 10, 50, 100
```

NOTEBOOK Statement

Syntax:

```
  NOTEBOOK   id, x, y, width, height[, style]
```

The NOTEBOOK statement creates a notebook control within the dialog window. This control is used to organize information on individual pages so that it can be located and displayed easily. The NOTEBOOK statement defines the identifier, position, dimensions, and attributes of a notebook control. The predefined class for this control is WC_NOTEBOOK. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_NOTEBOOK. You can use the bitwise OR (|) operator to combine styles.

Comments

The NOTEBOOK statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a notebook control at position (20, 20) within the dialog window. The notebook has a width of 200 character units and a height of 50 character units. Its resource identifier is 201. The tabs style BKS_ROUNDEDTABS specification overrides the notebook default style of square tabs. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define    IDC_NOTEBOOK     201
#define    IDD_NOTEBOOKDLG  503
DIALOG "Notebook", IDD_NOTEBOOKDLG, 11, 11, 420, 420, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    NOTEBOOK   IDC_NOTEBOOK, 20, 20, 200, 400, BKS_ROUNDEDTABS | WS_GROUP
  END
```

POINTER Statement

Syntax:

```
  POINTER pointer-id  [load-option]  [ mem-option] [codepage] filename
```

The POINTER statement defines a pointer resource for an application. A pointer resource, typically created by using the OS/2 Icon Editor, is a bit map defining the shape of the mouse pointer on the screen. The POINTER statement copies the pointer resource from the file specified in the filename field and adds it to the application's other resources. A pointer resource can be loaded from the executable file when needed by using the WinLoadPointer function.

You can provide any number of POINTER statements in a resource script file, but each statement must specify a unique pointer-id value.

pointer-id	Specifies the pointer-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinLoadPointer function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or	

	more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.	
filename	Specifies the name of the file containing the pointer resource. If the file is not in the current directory, filename must be preceded by a full path.	

Example

This example defines a pointer whose pointer identifier is 10. The pointer resource is copied from the file custom.cur.

```
POINTER 10 custom.cur
```

PRESPARAMS Statement

Syntax:

```
  PRESPARAMS presparam, value, presparam, value, ...
```

The PRESPARAMS statement defines presentation fields that customize a dialog box, menu, window, or control. PRESPARAMS data is a series of types and values. The window procedure of the dialog box, menu, window or control receives and processes this data when the item is created. The data for custom controls can be in any format.

presparam	Specifies a presentation-field type.
value	Specifies the presentation-field value.

Comments

PRESPARAMS is often used to supply data to control the appearance of the customized window when it is first created. For example, the PRESPARAMS statement may specify the colors to be used in the window.

Example

This example creates a menu resource with a menu identifier of 1. The PRESPARAMS statement specifies that the following three menu items be displayed in the 12-point Helvetica font.

```
MENU 1
BEGIN
    PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
    MENUITEM "New", 100
    MENUITEM "Open", 101
    MENUITEM "Save", 102
END
```

PUSHBUTTON Statement

Syntax:

```
  PUSHBUTTON text, id, x, y, width, height[, style]
```

The PUSHBUTTON statement creates a pushbutton control. The control is a round-cornered rectangle containing the given text. The control sends a message to its parent whenever the user chooses the control. The PUSHBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON and WS_TABSTOP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a pushbutton control that is labeled "OK."

```
PUSHBUTTON "OK", 101, 10, 10, 100, 100
```

RADIOBUTTON Statement

Syntax:

```
  RADIOBUTTON text, id, x, y, width, height[, style]
```

The RADIOBUTTON statement creates a radio-button control. The control is a small circle that has the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected. The RADIOBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_RADIOBUTTON.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.

style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a radio-button control that is labeled "Italic."

```
RADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

RCDATA Statement

Syntax:

```
  RCDATA resource-id
  BEGIN
  data-definition, data-definition   ...
      .
      .
      .
  END
```

The RCDATA statement defines a custom-data resource for an application. The custom data can be in whatever format the application requires. You can provide any number of RCDATA statements in a resource script file, but each statement must specify a unique resource-id value. A custom-data resource can be loaded from the executable file when needed by using the DosGetResource or DosGetResource2 functions with the RT_RCDATA resource type.

resource-id	Specifies the custom-data identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
data-definition	Specifies the custom data. The data may be simple expressions or strings.

Example

This example defines custom data that has a resource identifier of 5.

```
RCDATA 5
BEGIN
    "E. A. Poe", 1849, -32, 3L, 0x8000000l, 3+4+5
END
```

RCINCLUDE Statement

Syntax:

```
  RCINCLUDE filename
```

The RCINCLUDE statement causes RC to process the resource script file specified in the filename field along with the current resource script file. The contents of both files are compiled by RC and the results are placed in one binary resource file and/or executable file.

filename	Specifies the name of the resource script file to be included. If the file is not in the current directory, filename must be preceded by a full path.

Comments

RCINCLUDE statements are processed before any other processing is done, including preprocessing by RCPP.EXE, which removes comments, replaces values in the define directives, and so on.

When specifying a high performance file system (HPFS) file name on an RCINCLUDE statement, enclose the path and file name in double quotes; for example:

```
   RCINCLUDE "d:\project\long dialog.dlg"
```

Double quotes enables the Resource Compiler to recognize a name containing embedded blank characters.

Example

This example includes the file DIALOGS.RC as part of the current resource script file.

```
RCINCLUDE dialogs.rc
```

RESOURCE Statement

Syntax:

```
  RESOURCE type-id resource-id [load-option] [mem-option]
           [code-page] filename

  or

  RESOURCE type-id resource-id [load-option] [mem-option]
           [code-page]
  BEGIN
  data-definition [, data-definition]...
  ...
  END
```

The RESOURCE statement defines a custom resource for an application. A custom resource can be any data in any format. The RESOURCE statement copies the custom resource from the specified file or inline data, and adds it to the application's other resources. A custom resource can be loaded from the executable file when needed by using the DosGetResource or DosGetResource2 function and specifying the resource's type and resource identifier.

The custom resource data can be defined in a separate file or as inline data in the input script. This is reflected in the two formats that can be used for this statement. The first format is used when the custom resource data is being read from a file. The second format is used when the data consists of a block of raw source data that is defined inline in the input script.

You can provide any number of RESOURCE statements in a resource script file, but each statement must specify a unique combination of type-id and resource-id values. That is, RESOURCE statements having the same type-id value are permitted as long as the resource-id value for each is unique.

type-id	Specifies the custom-resource type. This value must be an integer in the range 256 through 65535, or a simple expression that evaluates to a value in that range. (Values 0 through 255 are reserved.)	
resource-id	Specifies the custom-resource identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.

codepage	Specifies a code page value. See CODEPAGE Statement. for a list of valid code pages.
filename	Specifies the name of the file containing the custom resource. If the file is not in the current directory, filename must be preceded by a full path.
data-definition	Specifies a custom data definition. The data can be a simple expression or a string. Integers can be specified in decimal, octal, or hexadecimal format. Data definitions in series on the same line are separated by commas. An integer specified without the suffix L must be in the range -32768 through 65535. An integer with an L suffix must be within the range -2147483648 through 4294967295. String data is specified withing quotes. **Note:** The Resouce Compiler does not append a null character to the end of these strings as it does for RCDATA blocks; any required null characters must be written as \0 within the data string.

Example

This example defines a custom resource whose type identifier is 300 and whose resource identifier is 14. The custom resource is copied from the file CUSTOM.RES.

```
RESOURCE 300 14 custom.res
```

RTEXT Statement

Syntax:

```
  RTEXT text, id, x, y, width, height[, style]
```

The RTEXT statement creates a right-aligned text control. The control is a simple rectangle displaying the given text right-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The RTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for the control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_RIGHT, and WS_GROUP.

text	Specifies text that is right-aligned in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a right-aligned text control that is labeled "Filename."

```
RTEXT "Filename", 101, 10, 10, 100, 100
```

SLIDER Statement

Syntax:

```
  SLIDER   id, x, y, width, height[, style]
```

The SLIDER statement creates a slider control within the dialog window. This control lets the user set, display, or modify a value by moving a slider arm along a slider shaft. The SLIDER statement defines the identifier, position, dimensions, and attributes of a slider control. The predefined class for this control is WC_SLIDER. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value can be any combination of the styles specified for WC_SLIDER. You can use the bitwise OR (|) operator to combine styles.

Comments

The SLIDER statement is only used in a DIALOG or WINDOW statement.

Example

This example creates a slider control at position (40, 30) within the dialog window. The slider has a width of 120 character units and a height of 2 character units. Its resource identifier is 101. The style specification SLS_BUTTONSLEFT adds buttons to the left of the slider shaft. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, though only the latter is specified.

```
#define    IDC_SLIDER       101
#define    IDD_SLIDERDLG    502
DIALOG "Slider", IDD_SLIDERDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    SLIDER   IDC_SLIDER, 40, 30, 120, 16, SLS_BUTTONSLEFT | WS_VISIBLE
  END
```

SPINBUTTON Statement

Syntax:

```
  SPINBUTTON   id, x, y, width, height[, style]
```

The SPINBUTTON statement creates a spin button control within the dialog window. This control gives the user quick access to a finite set of data. The SPINBUTTON statement defines the identifier, position, dimensions, and attributes of a spin button control. The predefined class for this control is WC_SPINBUTTON. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and SPBS_MASTER.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.

height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value is any combination of the styles specified for WC_SPINBUTTON. You can use the bitwise OR (|) operator to combine styles.

Comments

The SPINBUTTON statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a spin-button control at position (80, 20) within the dialog window. The spin button has a width of 60 character units and a height of 3 character units. Its resource identifier is 302. The style specification SPBS_NUMERICONLY creates a control which accepts only the digits 0-9 and virtual keys. The default styles SPBS_MASTER, WS_TABSTOP, and WS_VISIBLE are all in effect, though only WS_TABSTOP is specified.

```
#define    IDC_SPINBUTTON   302
#define    IDD_SPINDLG    502
DIALOG "Spin button", IDD_SPINDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    SPINBUTTON  IDC_SPINBUTTON, 80, 20, 60, 24, SPBS_NUMERICONLY | WS_TABSTOP
  END
```

STRINGTABLE Statement

Syntax:

```
  STRINGTABLE  [load-option] [mem-option] [codepage]
  BEGIN
  string-id string-definition
      .
      .
      .
  END
```

The STRINGTABLE statement creates one or more string resources for an application. A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using the WinLoadString or with DosGetResource with the RT_STRING resource type. RT_STRING resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential identifier. The resource string identifier is not necessarily the same as the identifier specified when using DosGetResource. The formula for calculating the identifier of the resource bundle, for use in DosGetResource, is as follows:

```
bundle ID = (id / 16) +1
```

where id is the string ID assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated identifier), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of STRINGTABLE statements in a resource script file. The compiler treats all the strings from the various STRINGTABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:

PRELOAD	System loads the resource when the application starts.
LOADONCALL	System loads the resource when the application calls the WinLoadString function. This is the default option.

mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:

FIXED	System keeps the resource at a fixed memory location.
MOVEABLE	System moves the resource as necessary to compact memory.
DISCARDABLE	System discards the resource if it is no longer needed.

The default setting is MOVEABLE and DISCARDABLE.

code-page	Specifies a code page value. See CODEPAGE Statement for a list of valid code page values.
string-id	Specifies the character-string identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. The value can be specified in decimal or hexadecimal notation. Each string identifier in a resource script file must be unique.
string-definition	Specifies a character string. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice.

Comments

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

STRINGTABLE
BEGIN
    IDS_HELLO   "Hello"
    IDS_GOODBYE "Goodbye"
END
```

SUBITEMSIZE Statement

Syntax:

```
  SUBITEMSIZE  size
```

The SUBITEMSIZE statement specifies the size, in words, of each help subitem in a help subtable. The minimum size is two words, and each help subitem in a help subtable must be the same size. When used, the SUBITEMSIZE statement must appear after the HELPSUBTABLE statement and before the BEGIN keyword.

You do not need to use the SUBITEMSIZE statement if the help subitems are the default size (2).

size	Specifies the size of each help subitem. This value must be an integer.

Example

The SUBITEMSIZE statement in this example specifies that each HELPSUBITEM statement contains three words.

```
HELPSUBTABLE 1
SUBITEMSIZE 3
BEGIN
```

```
    HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU, 5
    HELPSUBITEM IDCLD_HELPMENU, IDHP_HELPMENU, 6
END
```

SUBMENU Statement

Syntax:

```
  SUBMENU text, submenu-id [, menuitem-style]
  BEGIN
  menuitem-definition
      .
      .
      .
  END
```

The SUBMENU statement creates a submenu for a given menu. A submenu is a vertical list of menu items from which the user can choose a command.

You can provide any number of SUBMENU statements in a MENU statement, but each SUBMENU statement must specify a unique submenu-id value. You can provide any number of menuitem-definition statements in the SUBMENU statement. These define the menu items (commands) in the menu. The order of the statements determines the order of the menu items.

text	Specifies the text of the submenu. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice. A tilde (~) character in the item name indicates that the following character is used as a mnemonic character for the item. When the menu is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the menu item by pressing the key corresponding to the underlined mnemonic character.
submenu-id	Specifies the submenu identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
menuitem-style	Specifies the submenu style. This value can be a combination of MIS_ values. For details on the MIS_ values, see MENUITEM Statement.
menuitem-definition	Specifies a PRESPARAMS or MENUITEM statement. You can use the PRESPARAMS statement to control the appearance of a submenu, such as the font and the foreground and background colors. If used, the PRESPARAMS statement must immediately follow the BEGIN keyword. For details about the PRESPARAMS statement, see PRESPARAMS Statement. The MENUITEM statement defines an individual command in the given menu. For details, see MENUITEM Statement.

Example

This example creates a submenu named Elements. Its identifier is 2. The submenu contains three menu items, which are created by using MENUITEM statements.

```
SUBMENU "Elements", 2
BEGIN
    MENUITEM "Oxygen", 200
    MENUITEM "Carbon", 201, , MIA_CHECKED
    MENUITEM "Hydrogen", 202
END
```

----

undef Directive

Syntax:

```
  undef name
```

This directive removes the current definition of the specified name. All subsequent occurrences of the name are processed without replacement.

name	Specifies the name to be removed. This value is any combination of letters, digits, and punctuation.

Example

This example removes the definitions for the names "nonzero" and "USERCLASS".

```
#undef     nonzero
#undef     USERCLASS
```

VALUESET Statement

Syntax:

```
  VALUESET   id, x, y, width, height[, style]
```

The VALUESET statement creates a value set control within the dialog window. This control lets a user select one choice from a group of mutually exclusive choices. The VALUESET statement defines the identifier, position, dimensions, and attributes of a value set control. The predefined class for this control is WC_VALUESET. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value is any combination of the styles specified for WC_VALUESET. You can use the bitwise OR (|) operator to combine styles.

Comments

The VALUESET statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a value set control at position (40, 40) within the dialog window. The value set control has a width of 220 character and a height of 20 character units. Its resource identifier is 302. The style specification VS_ICON creates a control to show items in icon form. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, though only WS_TABSTOP is specified.

```
#define    IDC_VALUESET     302
#define    IDD_VALUESETDLG  501
DIALOG "Value set", IDD_VALUESETDLG, 11, 11, 260, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    VALUESET  IDC_VALUESET, 40, 40, 220, 160, VS_ICON | WS_TABSTOP
  END
```

WINDOW Statement

Syntax:

```
  WINDOW text, id, x, y, width, height, class[, style[, framectl]]
         data-definitions
[ BEGIN
control-definition
    .
    .
    .
END ]
```

The WINDOW statement creates a window of the specified class. The statement defines the position and dimensions of the window relative to its parent window, as well as the window-box style. The WINDOW statement is typically used in a WINDOWTEMPLATE or FRAME statement.

Typically, only one WINDOW statement is used in a FRAME statement. It defines the client window belonging to the corresponding frame window. The optional BEGIN and END keywords enclose any CONTROL statements that are given with the window. CONTROL statements given in this manner represent child windows belonging to the window created by the WINDOW statement.

text	Specifies the window title if the style specifies a title bar. This field must contain zero or more characters enclosed in double quotation marks. The character values must be in the range 1 through 255. If a double quotation mark is required in the title, you must include the double quotation mark twice.
id	Specifies the window identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units. The position is relative to the origin of the parent window.
y	Specifies the y-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units. The position is relative to the origin of the parent window.
width	Specifies the width of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
class	Specifies the window class. This value can be one of the control classes specified in the "Control Classes" table in the *Presentation Manager Programmer Reference* or the name of the window class, enclosed in double quotation marks.
style	Specifies the window style. This value can be any of the window, dialog box, or frame styles specified.
framectl	Specifies the style of the frame controls belonging to the window. This value can be a combination of the styles specified in the table, "Frame-Control Styles." You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the window. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the window.

Comments

The WINDOW statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a WINDOW statement contains one or no such statements.

Example

This example creates a client window belonging to the frame window. The client window belongs to the "MyClientClass" window class and has the standard window identifier FID_CLIENT.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130,
             0, FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
```

```
END
```

WINDOWTEMPLATE Statement

Syntax:

```
  WINDOWTEMPLATE window-id [load-option] [mem-option] [code-page]
  BEGIN
  window-definition
      .
      .
      .
  END
```

The WINDOWTEMPLATE statement creates a window template. A window template consists of a series of statements that define the window identifier, load and memory options, window dimensions, and controls in the window. The window template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of window templates in a resource script file, but each template must have a unique window-id value.

window-id	Specifies the window identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinLoadDlg function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and
	DISCARDABLE.	
code-page	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.	
window-definition	Specifies a WINDOW statement. The statement defines the dimensions and style of the given window. For details about the statement, see WINDOW Statement.	

Comments

A WINDOWTEMPLATE statement can contain DIALOG, CONTROL, and WINDOW statements. Typically, only one WINDOW statement is used in the WINDOWTEMPLATE statement.

Resource Compiler Error Messages

The error messages produced by the resource compiler utility (RC) and its preprocessor are listed below.

RC Preprocessor Fatal Error Messages

Error Message Descriptions

C1012

bad parenthesis nesting - missing symbol
Explanation: You wrote an expression which was missing the given left or right parenthesis symbol.
Action: Rewrite the expression with balanced parentheses.

C1014

too many include files
Explanation: You might have tried to include a file recursively.
Action: Remove the include directive for any file which has already been included to the preprocessor.

C1015

cannot open include file 'filename'
Explanation: The preprocessor could not locate the given include file.
Action: If the include file is not in the current directory or in a directory named in the INCLUDE environment variable, or in a directory specified by a -i option, you must provide the full path and include file name.

C1016

#if[n]def expected an identifier
Explanation: You wrote an ifdef or ifndef directive with no macro name.
Action: Supply the missing name.

C1017

invalid integer constant expression
Explanation: You used an incorrect expression where an integer constant was expected.
Action: Supply a correct expression to the directive.

C1018

unexpected #elif
Explanation: You used a directive in an incorrect context.
Action: Correct the logic of the if directives.

C1019

unexpected #else
Explanation: You used a directive in an incorrect context.
Action: Correct the logic of the if directives.

C1020

unexpected #endif
Explanation: You used a directive in an incorrect context.
Action: Correct the logic of the if directives.

C1021

bad preprocessor command 'command'
Explanation: The given command is not a recognized directive. You might have misspelled the directive.
Action: Use the correct spelling for the directive.

C1022

expected #endif
Explanation: You wrote an if directive but omitted any endif.
Action: Supply the missing endif directive.

C1056

compiler limit 'name' out of macro expansion space
Explanation: The macro called name expanded to a length exceeding 2042 bytes.
Action: Revise your definition of the macro so that its value is shorter than the length limit.

C1065

compiler limit 'name' macro definition too big
Explanation: The macro called name expanded to a length exceeding 2042 bytes.
Action: Revise your definition of the macro so that its value is shorter than the length limit.

Resource Compiler Preprocessor Error Messages

C2001

newline in constant
Explanation: You wrote a string literal constant without the closing double quotation mark.
Action: Provide the ending double quote for the string.

C2004

expected defined(id)
Explanation: You wrote an if defined directive but omitted the macro name.
Action: Supply the missing macro name.

C2006

#include expected a file name,found text
Explanation: The preprocessor found the given text instead of an include file name.
Action Supply the correct include file name.

C2007

#define syntax
Explanation: The syntax of your define directive is incorrect.
Action: Use the correct form as described for the define directive.

C2014

preprocessor command must start as first non-white space
Explanation: You wrote a directive with text on the line before the number sign (#).
Action: Put the number sign at the beginning of a line.

RC Preprocessor warnings

C4005

name redefinition
Explanation: You attempted to redefine the macro name to a value different from its current definition.
Action: To use a different macro value, define it as a macro using another macro name.

C4067

unexpected characters following include directive -newline expected
Explanation: You specified a filename to an include directive without surrounding the name by double quotation marks or angle brackets.
Action: Use double quotation marks or angle brackets around the include filename.

C4067

unexpected characters following 'endif' directive - newline expected
Explanation: No characters should appear on the line of an endif or else directive after the directive keyword.
Action: Remove the extra characters.

C4067

unexpected characters following 'else' directive-newline expected.
Explanation: No characters should appear on the line of an endif or else directive after the directive keyword.
Action: Remove the extra characters.

C4067

unexpected characters following 'undef' directive - newline expected
Explanation: You can undefine only one macro on an undef directive. **Action:** Use separate undef directive to undefine more than one macro.

Accelerator type required (CHAR, SCANCODE, or VIRTUALKEY)
Explanation: An *acceloption* has not been specified in the accelerator table to define the type of accelerator. If the accelerator character code is something other than a single character or a character preceded by a caret (ˆ), an *acceloption* is required.
Action: Check accelerator table syntax.

BEGIN expected in accelerator table
Explanation: BEGIN keyword missing from accelerator table.
Action: Check syntax.

BEGIN expected in dialog or window template
Explanation: BEGIN keyword missing from dialog or window template.
Action: Check syntax.

BEGIN expected in menu
Explanation: BEGIN keyword missing from menu.
Action: Check syntax.

BEGIN expected in message table
Explanation: BEGIN keyword missing from message table.
Action: Check syntax.

BEGIN expected in RCData
Explanation: BEGIN keyword missing from RCData table.
Action: Check syntax.

BEGIN expected in String Table
Explanation: BEGIN keyword missing from string table.
Action: Check syntax.

Cannot re-use message constants
Explanation: Message identifier has been used more than once in message table.
Action: Check message table syntax.

Cannot re-use string constants
Explanation: Message identifier has been used more than once in string table.
Action: Check string table syntax.

Comma expected after item string
Explanation: A comma must be used to separate the menu item identifier and the menu item string.
Action: Check menu syntax.

Control character out of range (ˆA - ˆZ)
Explanation: Accelerator character codes that use the Ctrl key, and are therefore preceded by a caret (ˆ), must use alphabetic keys.
Action: Check accelerator table syntax.

END expected in dialog
Explanation: END keyword missing from dialog template.
Action: Check syntax.

END expected in menu
Explanation: END keyword missing from menu.
Action: Check syntax.

Error creating temp file
Explanation: Temporary files are created by the resource compiler during the compilation process.
Action: Check that there is sufficient disk space to run the resource compiler, and restart the resource compiler.

Expected comma in accelerator table
Explanation: Commas are used in the accelerator table to separate the accelerator key, the accelerator command, and the accelerator options.
Action: Check accelerator table syntax.

Expected ID value for menu item
Explanation: A selection identifier is needed for each item within a menu.
Action: Check menu syntax.

Expected menu string
Explanation: A character string should be specified in the menu definition to describe the menu selection.
Action: Check menu syntax. The string should be enclosed in double quotation marks.

Expected numeric command value

Explanation: A number should be used in the accelerator table to identify the message that is generated by an accelerator key.
Action: Check accelerator table syntax.

Expected numeric constant in message table

Explanation: The identifier that precedes a message definition must be an integer.
Action: Check message definition syntax.

Expected numeric constant in string table

Explanation: The identifier that precedes a string definition must be an integer.
Action: Check string definition syntax.

Expected numerical dialog constant

Explanation: Integers are required in dialog and window templates to specify the coordinates and dimensions of the dialog box.
Action: Check syntax of dialog box definition.

Expected string in message table

Explanation: A character string was not found in the message table.
Action: Check syntax. The string should be enclosed in double quotation marks.

Expected string in string table

Explanation: A character string was not found in the string table.
Action: Check string table syntax. The string should be enclosed in double quotation marks.

Expected string or constant accelerator command

Explanation: The accelerator character code is missing.
Action: Check accelerator table syntax.

File not found

Explanation: The resource compiler could not find the .RC or .RES file that you requested.
Action: Check that the file is in the current directory and check the path to the directory.

Illegal empty BEGIN/END block found, resource not written

Explanation: A BEGIN/END block with no DIALOG, CONTROL, or WINDOW statements in it was found in the dialog template.
Action: Delete unwanted BEGIN/END blocks.

Invalid accelerator

Explanation: The character code specified as an accelerator key must be a valid keyboard operation.
Action: Check accelerator key definition syntax.

Invalid accelerator option

Explanation: The accelerator option must be a valid keyword.
Action: Check syntax.

Invalid control character

Explanation: The accelerator key definition can include a caret (^) to specify that the key should be used with the Ctrl key.
Action: Check accelerator key definition syntax.

Invalid Type

Explanation: The resource type must be a valid keyword.
Action: Check resource definition syntax.

Non-numeric template ID in dialog or window template

Explanation: The resource identifier must be an integer.
Action: Check dialog or window template syntax.

Only one top level window allowed

Explanation: Only one DIALOG, CONTROL, or WINDOW statement is allowed within the dialog or window template.
Action: Check dialog or window template syntax.

Resource Type keyword expected

Explanation: The resource type must be specified in the resource script file.
Action: Check resource definition syntax.

String literal too long

Explanation: Strings cannot be longer than 255 characters.
Action: Edit the string.

Text string or ordinal expected in control

Explanation: A text string can be specified in the DIALOG statement of a dialog template to give it a title. If a title is not required, double quotation marks must be used with no characters between them (" ").
Action: Edit DIALOG statement.

Unbalanced parentheses

Explanation: The left and right parentheses have not been matched.
Action: Edit the parentheses.

Undefined keyword or key name

Explanation: An invalid keyword or key name has been used.
Action: Check syntax.

Unexpected end of file in string literal

Explanation: The double quotation marks have not been closed at the end of a character string.
Action: Edit the string.

Unexpected value in RCData

Explanation: The variable defined in RCData must be a string or a number.
Action: Check the RCData syntax.

Unknown dialog or window token

Explanation: The dialog and window templates must use only the DIALOG, WINDOW, or CONTROL keywords.

Action: Check the dialog or window template syntax.

Unknown menu sub type

Explanation: Items within a menu can be specified only with the MENUITEM and SUBMENU keywords.
Action: Check menu definition syntax.

Notices

Copyright Notices

Disclaimers

Trademarks
